IN THE SPECIFICATION

Please amend the title to read DIGITAL SIGNAL PROCESSING APPARATUS HAVING MULTIPLE EXECUTION UNITS UNDER COMBINED GLOBAL AND LOCAL CONTROL

Page 3, please replace the second full paragraph with the following:

[0009] FIGS. 1a and 1b shows a simple example of a digital signal processor (DSP) loop computing a vector product which well represents a wide class of DSP algorithms (e.g. FIR filtering). FIG. 1a shows the original C code which can be compiled into a generic assembly code of a generic DSP core which assembly code is shown is FIG. 1b.

Please replace the paragraph bridging pages 3 and 4 with the following:

[0012] The vector product of the computation shown in FIGS. 1a and 1b for a VLIW DSP would look like the code given in FIG. 3. Bundles are composed by instructions separated by commas, whilst the bundles themselves are separated by semicolons. Even if the number of bundles is less than the number of instructions in the original code (cf. FIG. 1b vs. FIG. 3), the number of basic instructions has increased; in fact, it is not always possible to find independent instructions to fill the bundles, and a so-called "no-operation" (nop) instruction is thus required.

Page 7, please replace the final paragraph with the following:

[0030] FIGS. 1a and 1b shows a simple example of DSP loop computing vector product, expressed as C code (a) and as generic assembly code (b);

Page 8, please replace the first paragraph with the following:

[0031] FIGS. 2a and 2b shows block diagrams of a standard DSP core (a) and of a modem VLIW DSP core (b);

Page 8, please replace the sixth and seventh paragraphs with the following:

[0036] FIGS. 7a and 7b shows an example of a process using local control alone which requires timing synchronization in the manner of VLIW DSP cores yet (a) and using local control and FIFO registers for moving synchronization on the data-flow so as to simplify the process definition and reduce the number of required instructions (b);

[0037] FIGS. 8a and 8b shows the vector product for an original standard DSP core (a) and a possible version of the same piece of code for a DSP using local control and FIFO registers (b); and

Page 9, please replace the second full paragraph with the following:

[0042] The adoption of local control may, however, require that instructions are executed in a particular order in time corresponding to the synchronization among instructions in the same bundle of VLIW DSP cores (cf. FIG. 7a). Therefore, all functional units or execution elements are involved in each loop. In order to relax such constraint, synchronization to data is deferred. The instruction in the process which is waiting for a new data is stalled only. In order to easily include such synchronization on data, added to the provision of local controls are first-in/first-out (FIFO) queues used in the manner of registers (referred to as $f in the example of FIGS. 7a and 7b instead of $r as for standard registers in the example of FIG. 3 and 6). An instruction writing in a FIFO register is stalled only if the FIFO is full while an instruction reading a FIFO register is stalled only if data is not available. In this way, as shown in FIG. 7b, instructions exchange data through the FIFOs, and no additional "nop" instruction is required in the process. Synchronization data allows processes to be executed out-of-order in the manner of a super-scalar processor.

Please replace the pargraph bridging pages 9 and 10 with the following:

[0043] FIGS. 8a and 8b illustrates a possible code for implementing the vector product loop in the original standard DSP core (a) and in DSP core using local control and FIFO registers (b). In accordance with FIG. 8a, each instruction would be coded in 32-bit. However, according to FIG. 8b the "define_process" directive specifies a 3-instruction process. The directive itself is 32-bit and the local control 12 (cf. FIG. 5) stores only its information which is 18-bit (instead of 96-bit which would be required according to FIG. 8a). The register holding address #b stores in its tag the information {$f3, Read, first_instruction } and so on. Of course, the size of the tag depends on how this information is coded and complex.

Page 10, please replace the final paragraph with the following:

[0045] As it becomes clear from FIG<u>S</u>. 8<u>a and 8b</u> when compared with FIG. 3 and 4, the final code is shorter than the original; it replaces the loop statement with a repeat one which defines the repeat body as process B. Due to both synchronization on data and local control, all functional units or execution elements free of processes, where a process is either completed or not used (as process C), transfer control to the fetch unit and then can execute the instructions subsequent to the loop itself in parallel with the loop itself. This is not possible in standard solutions (e.g. conventional VLIW DSP) where in fact the units not involved in computation are either stalled or executing "nop" operations in order to respect timing constraints.